dee/estt

Microprocessadores e Aplicações

Acetatos de apoio às aulas teóricas

Ana Cristina Lopes
Dep. Engenharia Electrotécnica
http://orion.ipt.pt anacris@ipt.pt

Interrupções e Temporizadores

1. Interrupções:
   (a) Introdução;
   (b) Registos envolvidos;
   (c) Programação em C.
   (d) Interrupções externas;

2. Temporizadores/Contadores - Timer0:
   (a) Introdução;
   (b) Registos envolvidos;
   (c) Modos de Operação;
   (d) Interrupção associada.

3. Exemplos.

Introdução

- O PIC18F458 tem múltiplas fontes de interrupção, permitindo ainda que cada interrupção seja classificada como de prioridade alta ou prioridade baixa;
- O vector de prioridade elevada está no endereço 000008h e o de prioridade baixa está em 000018h;
- Os eventos associados a interrupções definidas como de prioridade alta saltarão por cima de interrupções de prioridade baixa que poderão estar a decorrer;
- A operação e controlo das interrupções são feitos com a ajuda de 13 registos, designadamente:
  - RCON;
  - INTCON;
  - INTCON2;
  - INTCON3;
  - PIR1, PIR2, PIR3;
  - PIE1, PIE2, PIE3;
  - IPR1, IPR2, IPR3;

Introdução

Cada fonte de interrupção requer três bits de controlo da sua operação, nomeadamente:

- Flag que indica se o evento de interrupção ocorreu;
- Bit de permissão de interrupção que permite que o programa salte para o endereço do vector de interrupção assim que a flag ficar a 1;
- Bit de prioridade que indica se a interrupção é de prioridade baixa ou elevada.

A definição de uma prioridade para cada interrupção pode ou não ser permitida, dependendo tal do valor do bit IPEN do registo RCON.

Se a prioritização das interrupções for permitida, existem dois bits que podem permitir globalmente as interrupções, designadamente:

- Bit GIEH (bit 7 do registo INTCON) - quando está a 1 permite todas as interrupções de prioridade alta não mascaradas (com o bit de permissão correspondente a 1);
- Bit GIEL (bit 6 do registo INTCON) - quando está a 1 permite todas as interrupções não mascaradas que tenham o bit de prioridade correspondente a 0 - prioridade baixa.

Introdução

Se para um dado evento de interrupção se verificarem as seguintes condições:

1. Flag correspondente a 1;
2. Bit de permissão correspondente a 1;
3. Bit global de permissão de interrupções apropriado a 1;

A interrupção será vectorizada imediatamente para os endereços 000008h ou 000018h consoante a prioridade da interrupção.

Note-se que as interrupções podem ser desactivadas individualmente através dos bits de permissão correspondentes.

Introdução

Se o bit IPEN estiver a zero (estado por defeito), a propriedade de prioritização das interrupções é desactivado e as interrupções do PIC18 passam a ser compatíveis com as dos PICs de gama média.

Note-se que neste modo de operação (de compatibilidade com outros PICs de gama média) os bits de prioridade de cada interrupção não têm qualquer efeito. Nestas condições:

- O bit GIE (bit 7 de registo INTCON) permite, quando está a 1, todas as fontes de interrupção não mascaradas (com bit de permissão de interrupção a 1);
- O bit PEIE (bit 6 do registo INTCON) permite, quando está a 1, todas as interrupções de periféricos não mascaradas;

Neste modo de compatibilidade, todas as interrupções são vectorizadas para o endereço 000008h.

Introdução

Enquanto o sistema está a responder a um determinado evento de interrupção que tenha ocorrido, o bit de permissão global de interrupções é limpo, de modo a impedir que outras interrupções ocorram em simultâneo;

Caso o bit IPEN esteja a 0, será o bit GIE que é limpo.

Caso o bit IPEN esteja a 1, então as prioridades das interrupções estão a ser utilizadas, e será o bit GIEH ou GIEL que será limpo, consoante a interrupção seja de alta ou baixa prioridade.

No último caso, uma interrupção de prioridade alta pode interromper uma interrupção de prioridade baixa.

Introdução

O endereço de retorno após a rotina de interrupção estar executada é colocado na pilha.

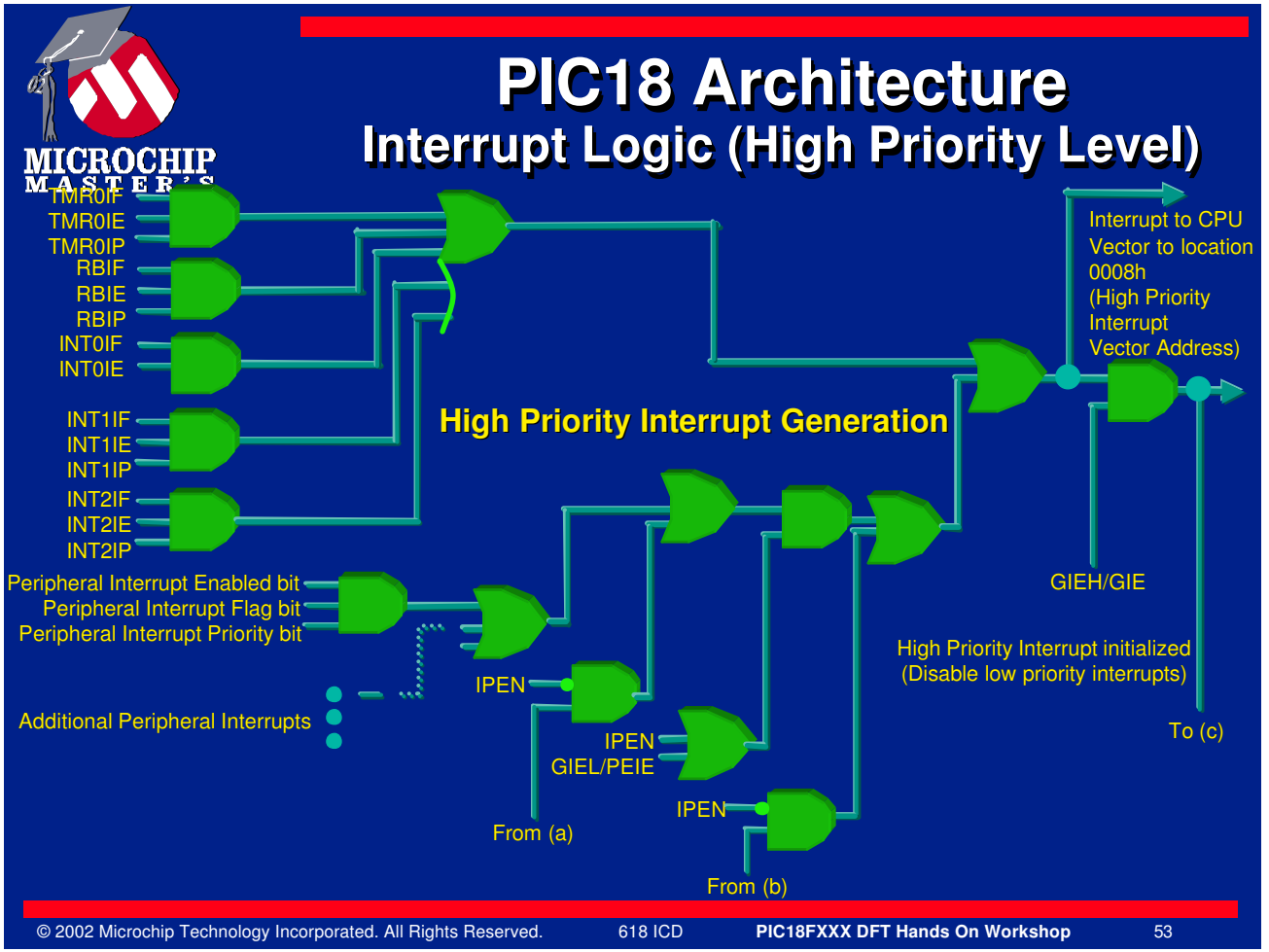
O PC é carregado com o endereço do vector de interrupção (000008h ou 000018h).

Uma vez dentro da rotina de serviço da interrupção (ISR - *Interrupt Service Routine*), o evento que desencadeou a interrupção pode ser determinado através de um teste realizado às diversas flags.

Note-se que as flags devem ser apagadas por software antes de se permitir globalmente as interrupções. A permissão global das interrupções é feita automaticamente assim que se retorna da rotina de interrupção. Por esse motivo, as flags devem ser apagadas dentro da rotina de interrupção.

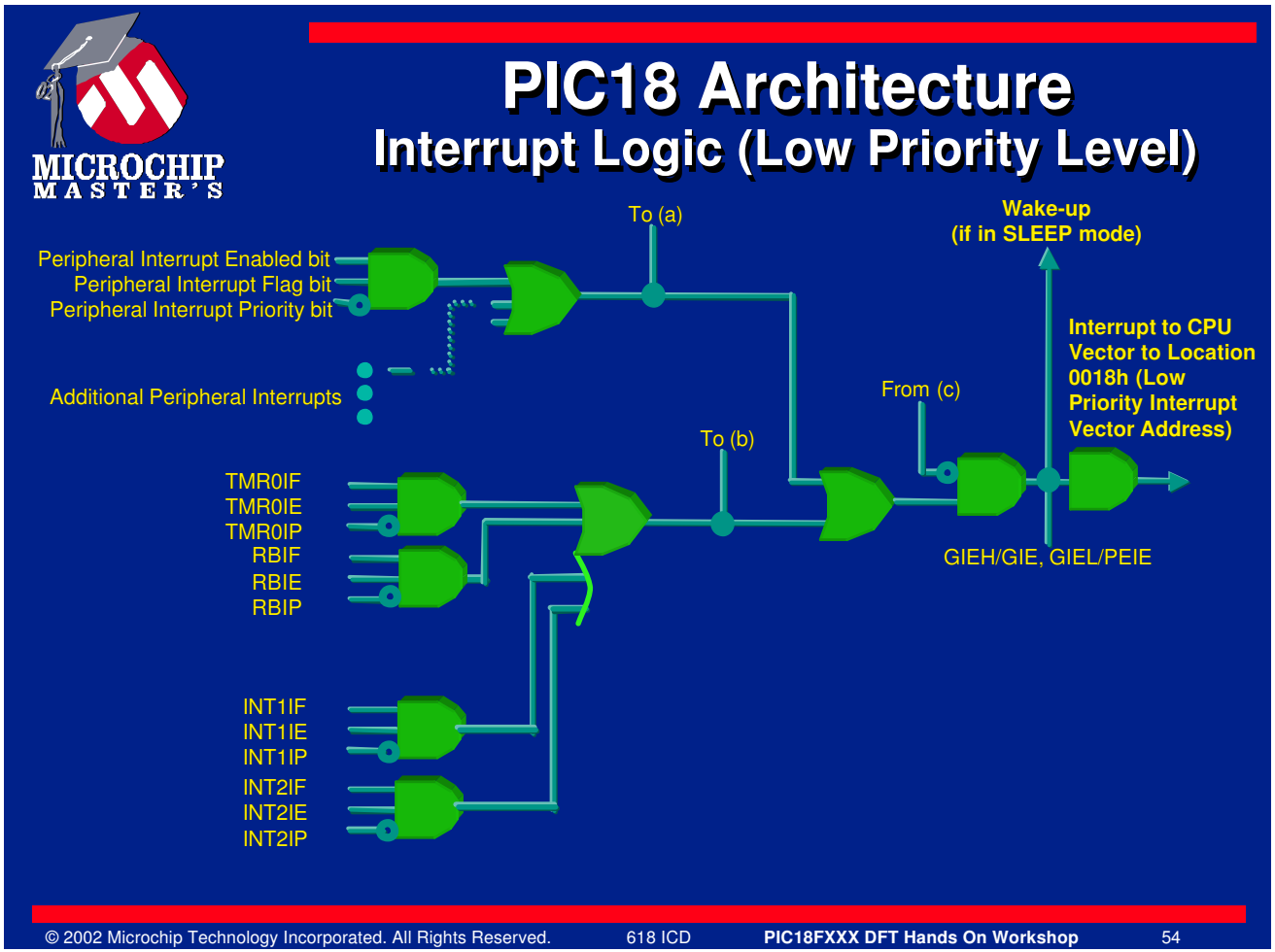
Introdução

A Figura seguinte mostra o circuito lógico responsável por gerar as interrupções de prioridade alta:



Introdução

A Figura seguinte mostra o circuito lógico responsável por gerar as interrupções de prioridade baixa:

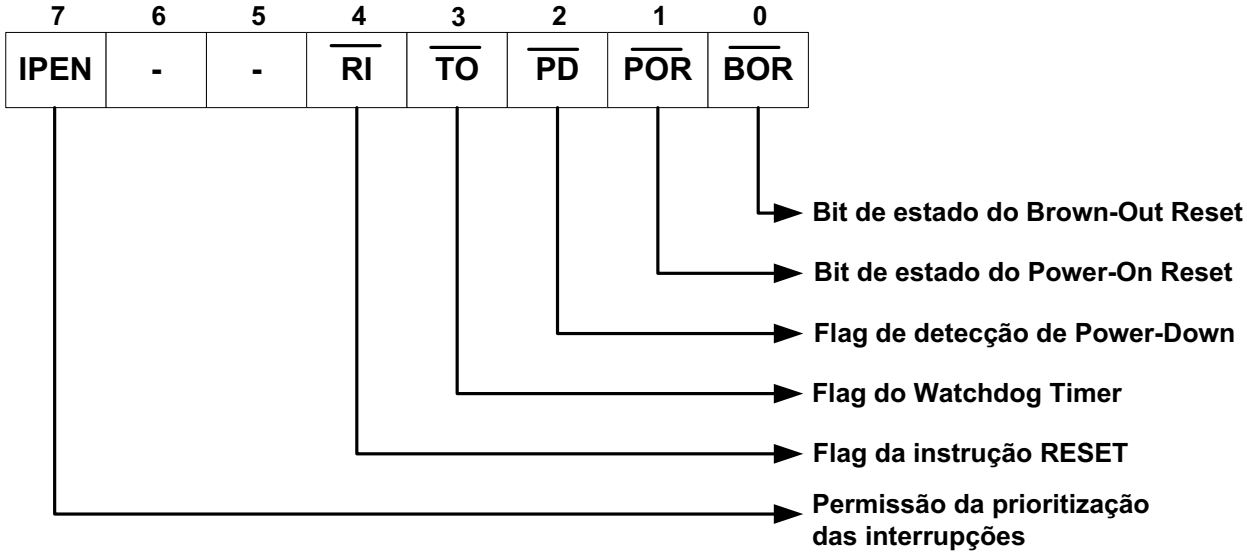


Registos Envolvidos - RCON

RCON - *Reset Control Register*

O registo RCON contém o bit IPEN que é usado para permitir a prioritização das interrupções. Os outros bits estão relacionados com outras características especiais do PIC18F458. A Figura seguinte mostra o conteúdo do registo RCON.

RCON – RESET CONTROL REGISTER



IPEN=1 – Permite prioritização das interrupções para nível alto ou baixo

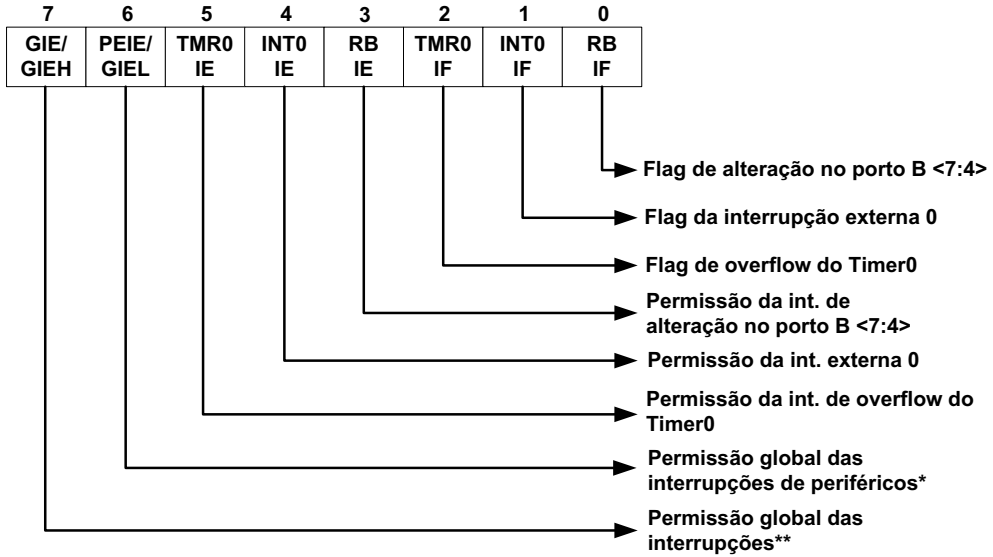
IPEN=0 - Não permite prioritização interrupção - compatível com famílias de PIC de gama média

Registos Envolvidos - INTCON

INTCON - *Interrupt Control Register*

O registo de escrita/leitura INTCON contém bits de permissão, prioridade e flags. Devido ao número de interrupções associadas ao PIC18F458 existem três registos de controlo de interrupções, designadamente INTCON, INTCON2 e INTCON3. A Figura seguinte mostra os conteúdos do registo INTCON.

INTCON – INTERRUPT CONTROL REGISTER



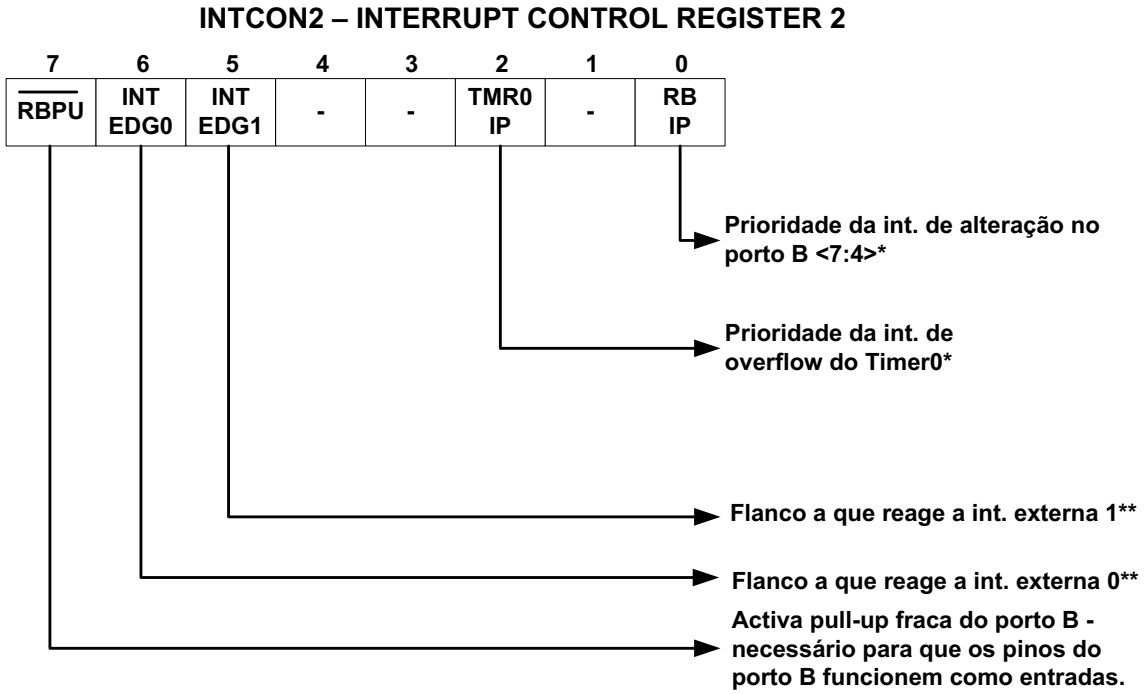
* Se IPEN = 0 permite todas as interrupções associadas a periféricos;
Se IPEN = 1 permite todas as interrupções de prioridade baixa.

** Se IPEN = 0 permite todas as interrupções;
Se IPEN = 1 permite todas as interrupções de prioridade alta.

Registos Envolvidos - INTCON2

INTCON2 - *Interrupt Control Register 2*

A Figura seguinte mostra os conteúdos do registo INTCON2.



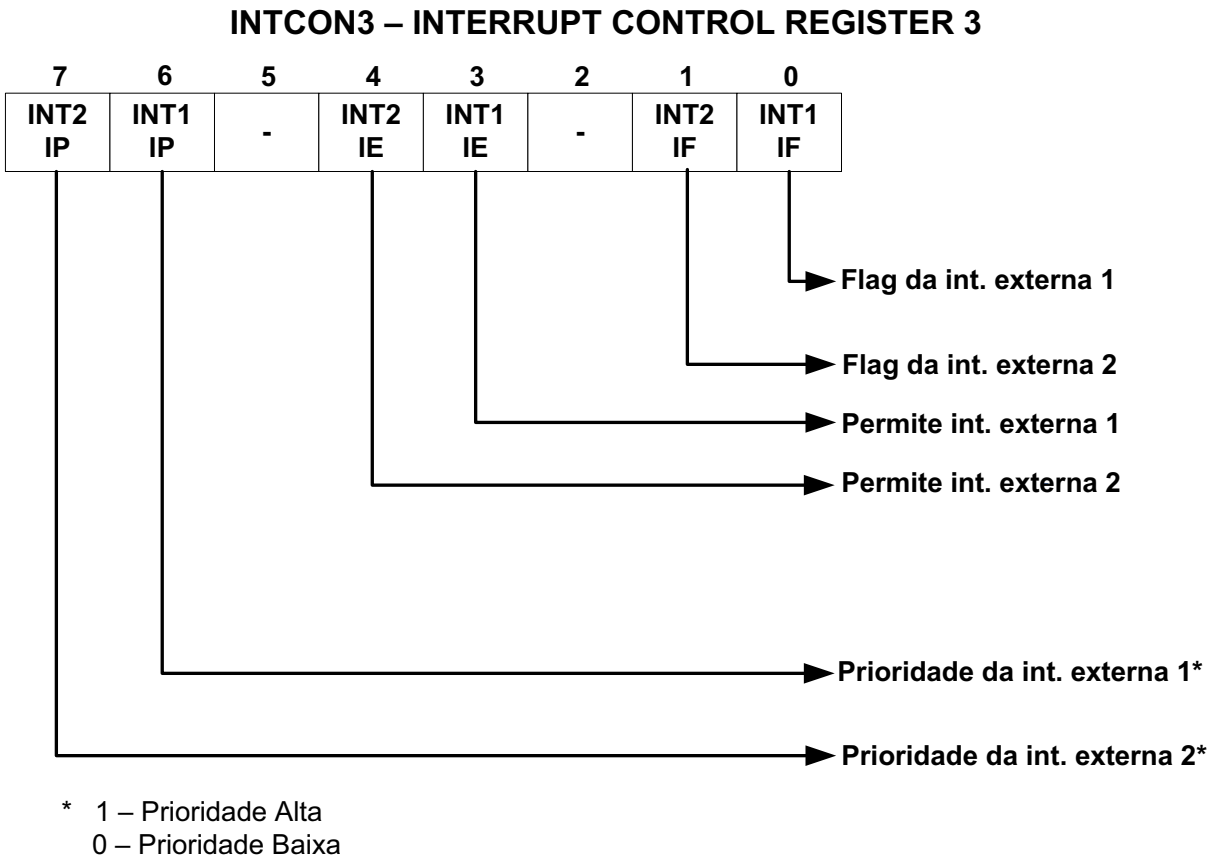
* 1 – Prioridade Alta
0 – Prioridade Baixa

** 1 – Flanco ascendente
0 – Flanco descendente

Registos Envolvidos - INTCON3

INTCON3 - *Interrupt Control Register 3*

A Figura seguinte mostra os conteúdos do registo INTCON3.



* 1 – Prioridade Alta
0 – Prioridade Baixa

Registos Envolvidos - Periféricos

Existem mais 9 registos associados às interrupções dos periféricos:

- Os registos PIR (PIR1, PIR2 e PIR3) - *peripheral Interrupt Request (Flag) Register 1, 2 e 3*- estes registos contém as flags associadas às interrupções de cada periférico (a necessidade de três registos desta natureza advém da existência de múltiplas fontes de interrupção de periféricos);
- Os registos PIE (PIE1, PIE2 e PIE3) - *peripheral Interrupt Enable Register 1, 2 e 3*- estes registos contém os bits de permissão de cada interrupção de periférico;
- Os registos IPR (IPR1, IPR2 e IPR3) - *peripheral Interrupt Priority Register 1, 2 e 3*- estes registos contém a prioridade (1 - alta ou 0 - baixa) de cada interrupção de periférico;

Programação em C

#pragma interrupt *fname* declara uma função que será uma rotina de serviço de interrupção (ISR - *Interrupt Service Routine*) de prioridade alta.

#pragma interruptlow *fname* declara uma função que será uma rotina de serviço de interrupção de prioridade baixa.

Sempre que uma interrupção ocorre:

- É suspensa a execução de qualquer aplicação que esteja a correr;
- A informação respeitante ao contexto actual é guardada;
- O controlo é transferido para uma ISR, de modo a que o evento que desencadeou a interrupção seja processado.

Programação em C

- O contexto mínimo que é guardado e restituído devido à ocorrência de uma interrupção é:
  - WREG;
  - BSR;
  - STATUS.
- As interrupções de prioridade alta utilizam os registos sombra - *shadow registers* - para guardar o contexto mínimo;
- As interrupções de prioridade baixa utilizam a pilha para guardar o contexto mínimo.
- As interrupções de prioridade alta terminam com um retorno de interrupção rápido - *Fast Return form Interrupt*;
- As interrupções de prioridade baixa terminam com um retorno de interrupção normal - *Return from Interrupt*.

Programação em C - #pragmas

As ISR utilizam uma secção de dados temporal que é distinta da que é normalmente utilizada pelas funções em C.

Qualquer dado temporário que seja requerido durante o processamento da interrupção é colocado nesta secção e não corre o risco de ser sobreposto com dados temporários de outras funções ou de outras rotinas de interrupção.

As pragmas das interrupções permitem que os dados temporários resultantes da execução das rotinas de interrupção sejam definidos ou declarados.

Caso não se defina nenhuma secção com este fim, serão criadas variáveis temporárias na secção udata sob o nome de fname_temp. No exemplo seguinte as variáveis temporárias da ISR foo serão colocadas na secção udata designada por foo_tmp:

```
void foo(void);
...
#pragma interrupt foo
void foo(void);
{
Função da rotina de interrupção
}
```

Programação em C - Sintaxe

Directiva interrupt:

♯pragma interrupt *function-name* [tmp-section-name] [save=save-list];
prioridade alta

♯pragma interruptlow *function-name* [tmp-section-name] [save=save-list];
prioridade baixa

save-list:

save-specifier;

save-list, save-specifier;

save-specifier:

symbol-name;

section("section-name");

Programação em C

De seguida apresenta-se o significado dos elementos apresentados anteriormente:

- *function-name* - identificador - designa a função em C que serve como ISR;
- *tmp-section-name* - identificador - designa a secção onde serão colocados os dados temporários da ISR;
- *symbol-name* - identificador - designa a variável que será restituída após o processamento da interrupção;
- *section-name* - identificador - (com a excepção de que o primeiro caracter pode ser um .)- designa a secção para onde será restituído o programa após o processamento da interrupção.

Programação em C - ISR

Uma ISR do MPLAB C18 é como qualquer outra função em C, podendo ter variáveis locais e podendo aceder a variáveis globais.

No entanto, é preciso notar que uma ISR deve ser declarada sem parâmetros e sem valor de retorno.

Note-se que uma ISR é chamada em resposta a uma interrupção de hardware, sendo realizada assincronamente.

As variáveis globais que serão acedidas tanto por ISRs como por funções normais devem ser declaradas como *volatile*.

Uma ISR só deve ser invocada por interrupções de hardware e nunca por outras funções de C.

A ISR utiliza a instrução RETFIE (*RETurn From Interrupt*) para sair da rotina de serviço de interrupção e nunca uma instrução RETURN.

Note-se que a utilização da instrução RETFIE fora de contexto pode corromper os registos WREG, BSR e STATUS.

Programação em C - Vectores

A Figura seguinte mostra como devem ser vectorizadas as interrupções de prioridade alta e baixa. O MPLAB C18 não vectoriza a ISR automaticamente. Normalmente coloca-se uma instrução GOTO dentro do vector da interrupção, por forma a que o controlo seja transferido para a ISR.

```
#include <p18f458.h>
void low_isr(void);
void high_isr(void);
/*
* Para o PIC18 o vector de endereço da interrupção de baixa prioridade está
* em 00000018h. Este código provoca um salto para a função
* low_interrupt_service_routine que integra o código das
* interrupções de prioridade baixa.
*/
#pragma code low_vector=0x18
void interrupt_at_low_vector(void)
{
_asm GOTO low_isr _endasm
}
#pragma code /* retorno à secção code por defeito*/
```

Programação em C - Vectores


```
/*
* CONTINUAÇÃO
*/
#pragma interruptlow low_isr
void low_isr (void)
{
/* ... */
}
/*
* Para o PIC18 o vector de endereço da interrupção de alta prioridade está
* 0000008h. Este código provoca um salto para a função
* high_interrupt_service_routine que integra o código das
* interrupções de prioridade alta.
*/
#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
_asm GOTO high_isr _endasm
}
#pragma code /* retorno à secção code por defeito */
#pragma interrupt high_isr
void high_isr (void)
{
/* ... */
}
```

Programação em C - Guardar contexto

Para além do contexto básico que é guardado por defeito existe a cláusula save= que permite que outros símbolos sejam guardados e restituídos.

Para guardar uma variável global definida pelo utilizador designada por *myint*, a directiva pragma deve ser utilizada da seguinte forma:

```
#pragma interrupt high_interrupt_service_routine save=myint
```

Também é possível guardar secções de dados. Considere-se uma secção de dados designada por mydata, neste caso a directiva pragma deve ser utilizada da seguinte forma:

```
#pragma interrupt high_interrupt_service_routine save=section("mydata")
```

Se uma ISR chama uma outra função, a secção de dados temporária dessa função (designada por ".tmpdata) deve ser guardada da seguinte forma:

```
#pragma interrupt high_interrupt_service_routine save=section(".tmpdata")
```

Programação em C - Guardar contexto

Numa forma resumida, a necessidade de adicionar save=[symbol or section] deriva de quatro razões principais:

- Aceder a um índice de um array calculado na ISR;
- Chamar outras funções dentro da ISR;
- Realizar operações matemáticas complicadas (*, / e vírgula flutuante);
- Aceder a variáveis da ROM.

A Tabela seguinte mostra as regras básicas para guardar o contexto durante o processamento da ISR.

Programação em C - Guardar contexto

Comportamento da ISR	Symbol ou section adicionado à ISR save-list
Chamar funções que também são chamadas no programa principal	section(".tmpdata"), PROD
Aceder a valores da memória de programa, tais como arrays, declarados como ROM-data	TABLPTR, TABLAT
Realizar uma multiplicação ou aceder a um índice de um array calculado na ISR	PROD
Realização de divisões, de multiplicações de 16 bits ou superior, operações com números de vírgula flutuante e outras operações científicas	section("MATH_DATA")

Exemplo: Aceder a um índice de um array calculado na ISR e executar uma divisão dentro da ISR:

```
#pragma interrupt sample_adc save=PROD, section("MATH_DATA")
```

Interrupções Externas

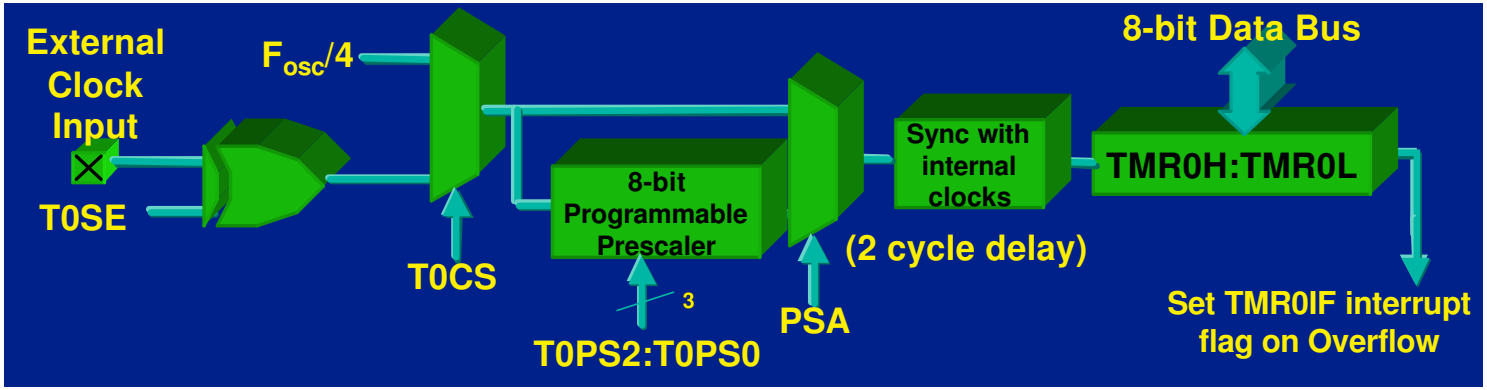
- As interrupções externas estão associadas aos pinos RB0/INT0, RB1/INT1 e RB2/CANTX/INT2;
- As interrupções são desencadeadas no flanco ascendente do sinal injectado nas entradas correspondentes, caso o bit INTEDGx do registo INTCON2 esteja a 1; caso contrário as interrupções são desencadeadas no flanco descendente;
- Assim que ocorra uma transição válida no pino RBx/INTx, a flag INTxIF correspondente é colocada a 1
- A interrupção é desactivada colocando a flag correspondente a zero;
- Note-se que a flag correspondente tem de ser limpa por software dentro da ISR e antes que a permissão de todas as interrupções ocorra novamente;
- A prioridade das interrupções INT1 e INT2 é determinada pelo valor contido nos bits INT1IP e INT2IP do registo INTCON3 (0 - prioridade baixa e 1 - prioridade alta);
- Não existe bit de prioridade associado à INT0, sendo esta sempre de prioridade alta.

Timer0

O módulo do Timer0 tem as seguintes características:

- Temporizador/contador de 16 ou 8 bits (seleccionável por software);
- Permite leitura e escrita;
- Prescaler de 8 bits programável por software;
- Fonte de relógio externa ou interna;
- Interrupção desencadeada por overflow de FFh para 00h (8 bits) ou de FFFFh para 0000h (16 bits);
- Permite a selecção de flanco ascendente ou descendente quando a fonte de relógio é externa.

A Figura seguinte mostra o diagrama de blocos respeitante ao Timer0.

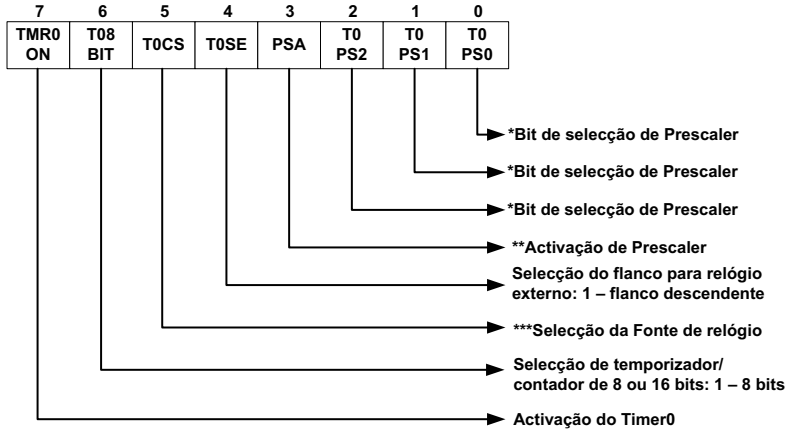


Timer0

O registo TCON é o registo que controla a operação do Timer0. A Figura seguinte mostra os conteúdos do registo TCON (por defeito todos os bits estão a 1).

T0CON – TIMER0 CONTROL REGISTER

7	6	5	4	3	2	1	0
TMR0 ON	T08 BIT	T0CS	T0SE	PSA	T0 PS2	T0 PS1	T0 PS0



* T0SP2:T0SP1: bits de selecção Prescaler do Timer0:
- 111 = 1:256
- 110 = 1:128
- 101 = 1:64
- 100 = 1:32
- 011 = 1:16
- 010 = 1:8
- 001 = 1:4
- 000 = 1:2

** Se PSA = 1 o Prescaler não está activo e a entrada de relógio do Timer0 passa por cima do Prescaler;
Se PSA = 0 o Prescaler fica activo e a entrada de relógio do Timer0 provém da saída do Prescaler.

*** Selecção da fonte de relógio: se T0CS = 1 transição no pino de entrada T0CKI (modo contador); se T0CS = 0 ciclo de instrução interno (modo temporizador).

Timer0

Modo de operação do Timer0:

- O Timer0 opera como temporizador se o bit T0CS estiver a zero. Quando o Prescaler não está activo tal implica que o registo do Timer0 (TMR0) sofre um incremento por ciclo de instrução;
- Se o bit T0CS estiver a 1 o Timer0 funciona como contador, tal implica que o registo do Timer0 (TMR0) será incrementado em cada flanco ascendente/descendente da entrada colocada no pino RA4/T0CKI;
- Assim que ocorre um transbordo do Timer0 a interrupção do timer0 é desencadeada;
- A flag do Timer0 - TMR0IF tem de ser limpa antes das interrupções voltarem a ser permitidas;
- Em C: utilização da função OpenTimer0(parâmetros).

Exemplo: Configurar o Timer0 para ser temporizador de 16 bits e para que ocorra uma interrupção assim que haja um overflow.

```
OpenTimer0 (TIMER_INT_ON & T0_SOURCE_INT & T0_16BIT);
```

Funções dos Timers

Função OpenTimerX (X = 0,1,2,3):

OpenTimer0:

Função: Configurar timer0.

Include: timers.h

Protótipo: void OpenTimer0(unsigned char config);

Argumentos: config;

Consiste numa máscara que é criada para realizar um operação AND (&) com os valores de cada uma das categorias que se encontram listadas de seguida.

Estes valores estão definidos no ficheiro timers.h.

Funções dos Timers

Definição dos argumentos:

- Permitir ou não a interrupção do Timer0:
  - permite interrupção - TIMER_INT_ON;
  - não permite interrupção - TIMER_INT_OFF;
- Dimensão do Timer0:
  - modo 8 bits - T0_8BIT;
  - modo 16 bits - T0_16BIT;
- Fonte de relógio:
  - Fonte externa - T0_SOURCE_EXT; neste caso o flanco é seleccionado da seguinte forma:
    - flanco ascendente - T0_EDGE_RISE;
    - flanco descendente - T0_EDGE_FALL;
  - Fonte interna - T0_SOURCE_INT.

Funções dos Timers

Função CloseTimerX (X = 0,1,2,3):

CloseTimer0:

Função: Desactivar o timer0.

Include: timers.h

Protótipo: void CloseTimer0(void);

Argumentos: config;

Nota: esta função também desactiva a interrupção associada ao Timer0.

Exercício

Realize os seguintes exercícios:

1. Realize de novo o programa da aula anterior, mas agora em vez de utilizar a função delay deve utilizar o timer0 e a interrupção que lhe está associada.

2. Faça um programa que acenda 8 leds ligados ao porto B da seguinte forma: 0000 0001, 0000 0011, 0000 0111, ...1111 1111, para tal deve utilizar a ISR associada ao timer0.

Em qualquer um dos casos utilize o valor máximo do timer0 sem prescaler - 0,0065535seg.